SyncML Sync Protocol, version 1.0.1

Abstract

This specification defines synchronization protocol between a SyncML client and server in form of message sequence charts. It specifies how to use the SyncML Representation protocol so that interoperating SyncML client and server solutions are accomplished.

Consortium

The following companies are sponsors in the SyncML initiative:
Ericsson
IBM
Lotus
Motorola
Nokia
Palm, Inc.
Matsushita Communications Industrial Co., Ltd.
Psion
Starfish Software

Revision History

Revision	Date	Comments
0.9	2000-05-31	0.9 release
1.0a	2000-08-31	Authentication procedures added, busy signaling generalized, multiple message per package functionality specified, Update command renamed to Replace, Alert codes modified, editorial changes.
1.0b	2000-11-07	Sync Anchors chapter updated, error cases fixed, slow sync chapter fixed, the sync alert chapter updated, examples updated
1.0	2000-12-07	The candidate version for the final release. The authentication example fixed. The device capabilities and the requirement for Get operation changed. Binary example updated. Examples updated to match with changes in the DevInf and MetInf specifications
1.0.1	2001-05-28	Incorporated Erratas
1.0.1	2001-05-03	Fixed copyright dates.

Copyright Notice

Table of Contents

1 Introduction

The purpose of this specification is to define a synchronization protocol using the SyncML Representation protocol [1]. This protocol is called the SyncML Sync Protocol. This specification defines the protocol for different sync procedures, which can occur between a SyncML client and a SyncML server, in the form of message sequence charts (MSC's). The specification covers the most useful and common synchronization cases (Chapters 4-8).

1.1 SyncML Framework

This specification can be implemented by using the SyncML interface from the SyncML Framework (See Figure 1). Not all the features of the SyncML Interface are required to comply with this specification.

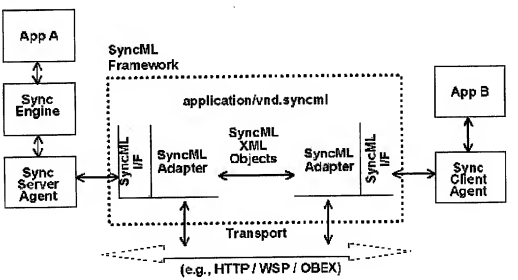

Figure 1 SyncML Framework

The application "A" depicts a networked service that provides data synchronization service for other applications, in this case application "B", on some networked device. The service and device are connected over some common network transport, such as HTTP.

In the figure above, the 'Sync Engine' functionality is completely placed onto the SyncML server side even if some SyncML client implementations may in practice provide some sync engine functionality, too. The 'Sync Server Agent' and the 'Sync Client Agent' use the protocol defined in this specification and the representation protocol [1] offered by the SyncML interface ('SyncML I/F') [2] to communicate with each other.

1.2 Device Roles

Figure 2 depicts a synchronization example in which a mobile phone acts as a SyncML client and a server acts as a SyncML server. The SyncML client sends SyncML message including the data modifications made in the client to the SyncML server. The server synchronizes the data (including

possible additions, replaces, and deletions) within the SyncML messages with data stored in the server. After that, the SyncML server returns its modifications back to the SyncML client.

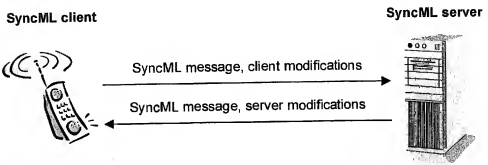

Figure 2 Synchronization Example with Mobile Phone and Server

The example presented the figure above is very simple. However, this example describes the roles of the devices in this specification. That is:

SyncML Client – This is the device that contains a sync client agent and that sends first its modifications to the server. The client must also be able to receive responses from the SyncML server. Although the SyncML client has always the role to send its modifications first, in some cases the server may have a role to initiate synchronization. The SyncML client is typically a mobile phone, PC, or PDA device.

SyncML Server – This is the device, which contains a sync server agent and sync engine, and which usually waits that the SyncML client starts synchronization and sends the clients modification to the server. The server is responsible for processing the sync analysis when it has received the client modifications. In addition, it may be able to initiate synchronization if unsolicited commands from the server to the client are supported on the transport protocol level. Typically, the SyncML server is a server device or PC.

1.3 Sync Types

This specification defines seven different sync types. These are introduced in Table 1.

Table 1 SyncML Sync Types

Sync Scenario	Description	Reference
Two-way sync	A normal sync type in which the client and the server exchange information about modified data in these devices. The client sends the modifications first.	Chapter 5
Slow sync	A form of two-way sync in which all items are compared with each other on a field-by-field basis. In practise, this means that the client sends all its data from a database to the server and the server does the sync analysis (field-by-field) for this data and the data in the server.	Chapter 5.5
One-way sync from client only	A sync type in which the client sends its modifications to the server but the server does not send its modifications back to the client.	Chapter 6

Refresh sync from client only	A sync type in which the client sends all its data from a database to the server (i.e., exports). The server is expected to replace all data in the target database with the data sent by the client.	Chapter 6.3
One-way sync from server only	A sync type in which the client gets all modifications from the server but the client does not send its modifications to the server.	Chapter 7
Refresh sync from server only	A sync type in which the server sends all its data from a database to the client. The client is expected to replace all data in the target database with the data sent by the server.	Chapter 7.5
Server Alerted Sync	A sync type in which the server to alerts the client to perform sync. That is, the server informs the client to starts a specific type of sync with the server.	Chapter 8

1.4 Symbols and conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interoperated as described in [RFC 2119].

Any reference to components of the Device Information DTD or XML snipets are specified in this `type face`.

1.4.1 MSC Notation

Notation used in the message sequence charts:

Box – Indicates the start of a procedure or an internal process in a device

Hexagon – Indicates a condition that is needed to start the transaction below this hexagon

Arrow – Represents a message, or transaction

5 Two-Way Sync

Two-way sync is a normal synchronization type in which the client and the server are required to exchange information about the modified data in these devices. The client is always the device which first sends the modifications. According to the information from the client, the server processes the synchronization request and the data from the client is compared and unified with the data in the server. After that, the server sends its modified data to the client device, which is then able to update its database with the data from the server.

In Figure 7, there is depicted the MSC of the client initiated two-way sync scenario.

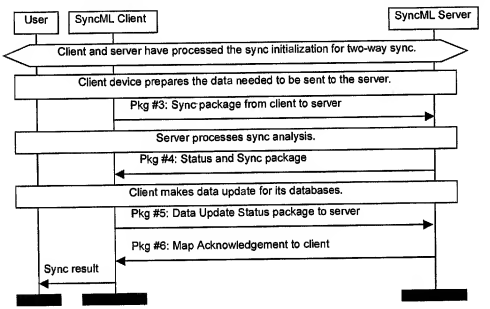

Figure 7 MSC of Two-Way Sync

The arrows in all figures in this document represent SyncML packages, which can include one or more messages. The package flow presented above is one SyncML session that means that all messages have the same SyncML session ID. The Session ID is same as used at the initialization.

The purpose and the requirements for each of the packages in the figure above are considered in the next sections.

Note. If the sync is done without a separate initialization (See Chapter 2.10), the number of a package in the figure may not describe the actual atomic number of a package in a synchronization session.

5.1 Client Modifications to Server

To enable sync, the client needs to inform the server about all client data modifications, which have happened since the previous sync package with modifications has been sent from the client to the

server[2] (Refer to the sync package, Pkg #3 in Figure 7). Any client modification, which is done after sending this package, MUST be reported to the server during the next sync session. It is not allowed to put them inside subsequent packages from the client to the server. The requirements for the sync package from the client to the server are following.

1. Requirements for the elements within the SyncHdr element.
   - The value of the VerDTD element MUST be '1.0'.
   - The VerProto element MUST be included to specify the sync protocol and the version of the protocol. The value MUST be 'SyncML/1.0' when complying with this specification.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging a sync session and traveling from the client to the server.
   - The Target element MUST be used to identify the target device and service.
   - The Source element MUST be used to identify the source device.
2. The Status MUST be returned for the Alert command sent by the client if requested by the server. This can be sent before Package #2 is completely received.
   - If the server is not authenticated to use the service, the sync type is wrong (e.g., slow sync needed), or some other error occurs, the client MUST return an error for that.
   - The next sync anchor of the server MUST be included in the Data element of Item (See 2.2.1).
3. If the server sent the device information to the client, the client SHOULD process the transmitted device information and the Status MUST be returned for that command if requested by the server. This can be sent before Package #2 is completely received.
4. If the server requested the device information of the client, the Results element MUST be returned. This can be sent before Package #2 is completely received.
   - The Type element of the MetaInf DTD MUST be included in the Meta element in the Results element to indicate that the type of the data is the type of the Device Information DTD.
   - The Source element in the Results element MUST have a value './devinf10'.
   - The Data element MUST be used to carry the device and service information of the client.
5. The Sync element MUST be included in SyncBody and the following requirements exist.
   - CmdID is required.
   - The response SHOULD be required for the Sync command.
   - Target is used to specify the target database.
   - Source is used to specify the source database.
   - The free memory SHOULD be specified inside the Meta element. The free memory can be either the free memory amount in the source database or the free memory amount on the client device (See Chapter 2.7). This information can only be sent at the first message belonging this package.

[2] These modifications include also modifications which have happened during the previous sync session after the client has sent its modifications to the server.

6. If there are modifications in the client, there are following requirements for the operational elements (e.g., Replace, Delete, and Add[3]) within the Sync element.
   - CmdID is required.
   - The response SHOULD be required for all these operations.
   - The Source element MUST be included to indicate the LUID (See Definitions) of the data item within the Item element.
   - The Type element of the MetaInf DTD MUST be included in the Meta element to indicate the type of the data item (E.g., MIME type). The Meta element inside an operation or inside an item can be used.
   - Data element MUST be used to carry data itself if the operation is not a deletion.
7. The Final element MUST be used for the message, which is the last in this package. After the server has received the final message of the package, it can complete the sync analysis and send its modifications back to client.

5.1.1 Example of Sending Modifications to Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.0</VerDTD>
      <VerProto>SyncML/1.0</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source><LocURI>IMEI:493005100592800</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>1</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>IMEI:493005100592800</TargetRef>
         <SourceRef> http://www.syncml.org/sync-server </SourceRef>
         <Data>212</Data> <!--Statuscode for OK, authenticated for session-->
      </Status>
      <Status>
         <CmdID>2</CmdID>
         <MsgRef>1</MsgRef><CmdRef>5</CmdRef><Cmd>Alert</Cmd>
         <TargetRef>./dev-contacts</TargetRef>
         <SourceRef>./contacts/james_bond</SourceRef>
         <Data>200</Data> <!--Statuscode for Success-->
         <Item>
            <Data>
               <Anchor xmlns='syncml:metinf'><Next>20000502T093223Z </Next></Anchor>
            </Data>
         </Item>
      </Status>
      <Sync>
         <CmdID>3</CmdID>
         <Target><LocURI>./contacts/james_bond</LocURI></Target>
         <Source><LocURI>./dev-contacts</LocURI></Source>
         <Meta>
            <Mem xmlns='syncml:metinf'>
               <FreeMem>8100</FreeMem>
```

[3] It is not required that the SyncML clients support the Add operation when sending modifications. They may use the Replace operation for additions and then, the receiving device must make addition if the UID of an object does not exist.

```
            <!--Free memory (bytes) in Calendar database on a device -->
            <FreeId>81</FreeId>
            <!--Number of free records in Calendar database-->
          </Mem>
        </Meta>
        <Replace>
          <CmdID>4</CmdID>
          <Meta><Type xmlns='syncml:metinf'>text/x-vcard</Type></Meta>
            <Item>
              <Source><LocURI>1012</LocURI></Source>
              <Data><!--The vCard data would be placed here.--></Data>
            </Item>
        </Replace>
      </Sync>
      <Final/>
    </SyncBody>
</SyncML>
```

5.2 Server Modifications to Client

The sync package (Refer Pkg #4 in Figure 7) to the client has the following purposes:

- To inform the client about the results of sync analysis.
- To inform about all data modifications, which have happened in the server since the previous time when the server has sent the modifications to the client.

Any server modifications, which are done after sending this package, MUST be reported to the client during the next sync session. It is not allowed to put them inside subsequent packages from the server to the client.

The requirements for messages within this sync package are following.

1. Requirements for the elements within the SyncHdr element.
   - The value of the VerDTD element MUST be '1.0'.
   - The value of the VerProto element MUST be 'SyncML/1.0'.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging a sync session and traveling from the server to the client.
   - The Target element MUST be used to identify the target device.
   - The Source element MUST be used to identify the source device and service.
2. The Status element MUST be included in SyncBody if requested by the client. It is now used to indicate the general status of the sync analysis and the status information related to data items sent by the client (e.g., a conflict has happened.). Status information for data items can be sent before Package #3 is completely received.
3. The Sync element MUST be included in SyncBody, if earlier there were no occurred errors, which could prevent the server to process the sync analysis and to send its modifications back to the client. For the Sync element, there are the following requirements.
   - CmdID is required.
   - The response can be required for the Sync command. (See the Caching of Map Item, Chapter 2.3.1)

- Target is used to specify the target database.
- Source is used to specify the source database.

4. If there is any modification in the server after the previous sync, there are following requirements for the operational elements (e.g., Replace, Delete, and Add[4]) within the Sync element.
   - CmdID is required.
   - The response can be required for these operations.
   - Source MUST be used to define the temporary GUID (See Definitions) of the data item in the server if the operation is an addition. If the operation is not an addition, Source MUST NOT be included.
   - Target MUST be used to define the LUID (See Definitions) of the data item if the operation is not an addition. If the operation is an addition, Target MUST NOT be included.
   - The Data element inside Item is used to include the data itself if the operation is not a deletion.
   - The Type element of the MetaInf DTD MUST be included in the Meta element to indicate the type of the data item (E.g., MIME type). The Meta element inside an operation or inside an item can be used.
5. The Final element MUST be used for the message, which is the last in this package.

5.2.1 Example of Sending Modifications to Client

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.0</VerDTD>
      <VerProto>SyncML/1.0</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>2</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>http://www.syncml.org/sync-server</TargetRef>
         <SourceRef>IMEI:493005100592800</SourceRef>
         <Data>200</Data>
      </Status>
      <Status><!--This is a status for the client modifications to the server.-->
      <CmdID>2</CmdID>
         <MsgRef>2</MsgRef><CmdRef>3</CmdRef><Cmd>Sync</Cmd>
         <TargetRef>./contacts/james_bond</TargetRef>
         <SourceRef>./dev-contacts</SourceRef>
         <Data>200</Data> <!--Statuscode for Success-->
      </Status>
      <Status>
         <CmdID>3</CmdID>
         <MsgRef>2</MsgRef><CmdRef>4</CmdRef><Cmd>Replace</Cmd>
         <SourceRef>1012</SourceRef>
         <Data>200</Data> <!--Statuscode for Success-->
      </Status>
      <Sync>
```

[4] It is not required that the devices support the Add operation. They may use the Replace operation for additions and then, the receiving device must make addition if the UID of an object does not exist.

```
        <CmdID>4</CmdID>
        <Target><LocURI>./dev-contacts</LocURI></Target>
        <Source><LocURI>./contacts/james_bond</LocURI></Source>
        <Replace>
           <CmdID>5</CmdID>
           <Meta><Type xmlns='syncml:metinf'>text/x-vcard</type></Meta>
           <Item>
              <Target><LocURI>1023</LocURI></Target>
              <Data><!--The vCard data would be placed here.--></Data>
           </Item>
        </Replace>
        <Add>
           <CmdID>6</CmdID>
           <Meta><Type xmlns='syncml:metinf'>text/x-vcard</type></Meta>
           <Item>
              <Source><LocURI>10536681</LocURI></Source>
              <Data><!--The vCard data would be placed here.--></Data>
           </Item>
        </Add>
     </Sync>
     <Final/>
  </SyncBody>
</SyncML>
```

5.3 Data Update Status from Client

The data update status package from the client to the server is needed to transport the information about the result of the data update on the client side. In addition, it is used to indicate the LUID's of the new data items, which have been added in the client, i.e., the Map operation for mapping LUID's and temporary GUID's is sent to the server.

Note. This package MAY NOT be sent if the server has indicated that it does not require a response to its last package to the client. If the client decides that it does not send this message, it MUST be able to cache the Map operations until the next synchronization will happen, when these Map operations can be sent to the server (See also Chapter 2.3.1). However, the client is always allowed to send this Data Update Status package to the server, even if the server has not requested a response.

The messages in this package have the following requirements.

1. Requirements for the elements within the SyncHdr element.
   - The value of the VerDTD element MUST be '1.0'.
   - The value of the VerProto element MUST be 'SyncML/1.0'.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging a sync session and traveling from the client to the server.
   - The Target element MUST be used to identify the target device and service.
   - The Source element MUST be used to identify the source device.
2. The Status element MUST be in SyncBody if requested by the server. It is used to indicate the results of data update in the client. Also, the status information related to the individual data items is transferred to the server. The status information for data items can be sent before Package #4 is completely received.

3. The Map element MUST be included in the SyncBody element if the client has processed any server additions to its database. For each database being synchronized, a separate Map operation or operations MUST be sent if any additions to a database is carried out. This command can be sent before Package #4 is completely received.
   - CmdID is required.
   - The Source and Target elements are required in the Map element.
   - The response is required to the Map operation.
   - The client has to return the client side IDs, i.e., LUID's and the server side IDs (temporary GUID's) for the data items within MapItem elements.
4. The Final element MUST be used for the message, which is the last in this package.

5.3.1 Example of Data Update Status to Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.0</VerDTD>
      <VerProto>SyncML/1.0</VerProto>
      <SessionID>1</SessionID>
      <MsgID>3</MsgID>
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source><LocURI>IMEI:493005100592800</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>2</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>IMEI:493005100592800</TargetRef>
         <SourceRef> http://www.syncml.org/sync-server </SourceRef>
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>2</CmdID>
         <MsgRef>2</MsgRef><CmdRef>4</CmdRef><Cmd>Sync</Cmd>
         <TargetRef>./dev-contacts</TargetRef>
         <SourceRef>./contacts/james_bond</SourceRef>
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>3</CmdID>
         <MsgRef>2</MsgRef><CmdRef>5</CmdRef><Cmd>Replace</Cmd>
         <TargetRef>1023</TargetRef>
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>4</CmdID>
         <MsgRef>2</MsgRef><CmdRef>6</CmdRef><Cmd>Add</Cmd>
         <SourceRef>10536681</SourceRef>
         <Data>200</Data>
      </Status>
      <Map>
         <CmdID>5</CmdID>
         <Target><LocURI>./contacts/james_bond</LocURI></Target>
         <Source><LocURI>./dev-contacts</LocURI></Source>
         <MapItem>
            <Target><LocURI>10536681</LocURI></Target>
            <Source><LocURI>1024</LocURI></Source>
         </MapItem>
      </Map>
      <Final/>
```

```
  </SyncBody>
</SyncML>
```

5.4 Map Acknowledgement from Server

The Map Acknowledgement from the server to the client is needed to inform the client that the server has received the mapping information of the data items. This acknowledgement is sent back to the client even if there were no Map operations in last package from the client to the server.

The messages in this package have the following requirements.

1. Requirements for the elements within the SyncHdr element.
   - The value of the VerDTD element MUST be '1.0'.
   - The value of the VerProto element MUST be 'SyncML/1.0'.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging a sync session and traveling from the server to the client.
   - The Target element MUST be used to identify the target device.
   - The Source element MUST be used to identify the source device and service.
   - The response MUST NOT be required for this message.
2. The Status element(s) MUST be included in SyncBody. It is now used to indicate the status of the Map operation(s). This or these can be sent before Package #5 is completely received.
3. The Final element MUST be used for the message, which is the last in this package.

5.4.1 Example of Map Acknowledge

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.0</VerDTD>
      <VerProto>SyncML/1.0</VerProto>
      <SessionID>1</SessionID>
      <MsgID>3</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>3</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>http://www.syncml.org/sync-server</TargetRef>
         <SourceRef>IMEI:493005100592800</SourceRef>
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>3</MsgRef><CmdRef>5</CmdRef><Cmd>Map</Cmd>
         <TargetRef>./contacts/james_bond </TargetRef>
         <SourceRef>./dev-contacts</SourceRef>
         <Data>200</Data>
      </Status>
      <Final/>
   </SyncBody>
```

</SyncML>

5.5 Slow Sync

The slow sync can be desired for many reasons, e.g., the client or the server has lost its change log information, the LUID's have wrapped around in the client, or the sync anchors mismatch. The slow sync is a form of the two-way synchronization in which all items in one or more databases are compared with each other on a field-by-field basis. In practise, the slow sync means that the client sends all its data in a database to the server and the server does the sync analysis (field-by-field) for this data and the data in the server. After the sync analysis, the server returns all needed modifications back to the client. Also, the client returns the Map items for all data items, which were added by the server.

Because of many reasons to process the slow sync, it can be either the client or the server, which indicates a need for this. If the client does this, it specifies in the Alert command that the sync type is the slow sync. The Alert command MAY be the same as at the sync initialization or the similar Alert command MAY be included when Package #3 is sent. The value of the Alert code is 201.

If there is a need for the server to initiate the slow sync, it happens by including the Alert operation with the 201 alert code. This alert operation MUST be the Alert operation at the Sync Initialization (Refer Package #2). After the client has received the status and the Alert operation for the slow sync, sync can be thought to start as if the client were initiating the slow sync in Package #3. However, the client MUST NOT include the Alert command anymore if it was the server, which alerted the slow sync.

If the client or the server needs to initiate the slow sync after receiving the alert for the normal synchronization, they need to send back an error status for that Alert in addition the slow sync alert. The error code, which is used in this case, MUST be 508 (Refresh required). If the client has not used a separate synchronization initialization, as specified in Chapter 2.10, it MUST send all updates in the next message to the server after receiving the error status and the Alert for a slow sync.

After the server has sent the Sync Alert, and if the client does not agree with the sync anchor in that Alert, then the Client MUST start a slow sync. This is done by sending back a Status on that Alert with Refresh Required (508). In this same message, the client should start the slow sync. In this case, the client MUST NOT send another Alert to start the slow sync. Note that it is not necessary for the client to compare the sync anchor from the server.

If the devices are synchronizing with each other at the first time, the slow sync MUST be initiated.

5.6 Error Case Behaviors

In this chapter, the recommended behaviors are defined in the cases of different error types.

5.6.1 No Packages from Server after Initialization

If the client has sent its modifications to the server and it does not get the status associated with those modifications, the client MUST assume that the server has not received those client modifications. At the next time when synchronization is started, the modifications, to which the status was not received, MUST be sent to the server again.

5.6.2 No Data Update Status from Client

If the server has sent its modifications to the client and it does not get the status associated with those server modifications, the server MUST assume that the client has not received those server modifications. Thus, at the next time when synchronization is started, the server modifications in addition to new ones MUST be sent to the client.

5.6.3 No Data Map Acknowledge from Server

If the client has sent the Map operation(s) and it does not get any complete response to it, the client SHOULD assume that the server has not received the Map operation(s). Thus, the client SHOULD try to send the Map operation(s) again or at the next time when synchronization is started.

5.6.4 Errors with Defined Error Codes

If the device receives a defined error code [1], it MUST act according that error type.

6 One-Way Sync from Client Only

The one-way sync from the client only is the sync type in which the client sends all modifications to the server but the server does not send its modifications back to the client. Thus, after this type of sync, the server includes all modified data from the client but the client does not know about modifications in the server. In Figure 8, there is depicted the MSC for this scenario.

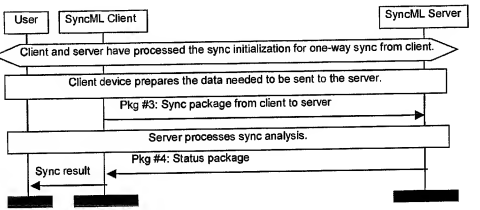

Figure 8 MSC of One-Way Sync from Client only

The package flow presented above is one SyncML session that means that all messages have the same SyncML session ID. The Session ID is same as used at the initialization.

The purpose and the requirements for each of package in the figure above are considered in the next sections.

Note. If the sync is done without a separate initialization (See Chapter 2.10), the number of a package in the figure may not describe the actual atomic number of a package in a synchronization session.

6.1 Client Modifications to Server

To initiate the sync, the client needs to inform the server about all client data modifications, which have happened since the previous sync[5] (Refer to the sync package, Pkg #3 in Figure 8). Any client modification, which is done after sending this package, MUST be reported to the server during the next sync session. It is not allowed to put them inside subsequent packages from the client to the server. The requirements for the sync package from the client to the server are the same as in Chapter 5.1.

6.2 Status from Server

The Status package (Refer Pkg #4) has a purpose of informing the client about the results of sync analysis. The requirements for the status package are following.

[5] These modifications include also modifications which have happened during the previous sync session after the client has sent its modifications to the server.

1. Requirements for the elements within the SyncHdr element.
   - The value of the VerDTD element MUST be '1.0'.
   - The value of the VerProto element MUST be 'SyncML/1.0'.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging a sync session and traveling from the server to the client.
   - Final MUST be used for the message, which is the last in this package.
   - The Target element MUST be used to identify the target device.
   - The Source element MUST be used to identify the source device and service.
2. The Status element MUST be included in SyncBody if requested by the client. It is now used to indicate the general status of the sync analysis and the status information related to data items sent by the client if this is necessary (e.g., a conflict has happened.). The status information for data items can be sent before Package #1 is completely received.

6.3 Refresh Sync from Client Only

The 'refresh sync from client only' is a synchronization type in which the client sends all its data from a database to the server (i.e., exports). The server is expected to replace all data in the target database with the data sent by the client. I.e., this means that the client overwrites all data in the server database.

This refresh sync is treated as a special case of the 'one-way sync from client only'. The only differences between this case and the normal 'one-way sync from client only' are:

1. At the initialization, the sync type (Alert code) MUST be used to indicate that the 'one-way refresh sync from client only' is required. The Alert code is 203.
2. In Package #3, the Sync element (Pkg #3) from the client to the server is required to include all data from the source database (client database).

6.4 Error Cases Behavior

In this chapter, the recommended behaviors of devices are defined in the cases of different error types.

6.4.1 No Packages from Server after Initialization

See Chapter 5.6.1.

6.4.2 Errors with Defined Error Codes

See Chapter 5.6.4.

7 One-Way Sync from Server only

This sync type is the case in which the client gets all modifications from the server but the client does not send its modifications to the server. Thus, after this type of sync, the client includes all modified data from the server but the server does not know about modifications in the client. In Figure 9, there is depicted the MSC for this scenario.

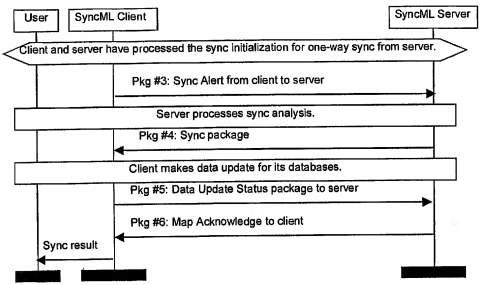

Figure 9 MSC of Sync from Server Only

The package flow presented above is one SyncML session that means that all messages have the same SyncML session ID. The Session ID is same as used at the initialization.

The purpose and the requirements for each of package in the figure above are considered in the next sections.

Note. If the sync is done without a separate initialization (See Chapter 2.10), the number of a package in the figure may not describe the actual atomic number of a package in a synchronization session.

7.1 Sync Alert to Server

The sync package (Pkg #3 in Figure 9) is very much similar to the package #3 in the two-way sync but any client modifications are not ever sent to server and the server is only asked to send its modifications to the client. The only difference from the requirements defined in Chapter 5.1 is:

1. Any client modifications are not included into the Sync element. It must be empty.

7.2 Server Modifications to Client

See Chapter 5.2.

7.3 Data Update Status from Client

See Chapter 5.3.

7.4 Map Acknowledge from Server

See Chapter 5.4.

7.5 Refresh Sync from Server Only

The 'refresh sync from server only' is a synchronization type in which the server sends all its data from a database to the client. The client is expected to replace all data in the target database with the data sent by the server. I.e., this means that the server overwrites all data in the client database.

This refresh sync is treated as a special case of the 'one-way sync from server only'. The differences between this case and the normal 'one-way sync from server only' are:

1. At the Sync Initialization (See Chapter 7.1), the value for the Alert code is 205.
2. In the Server Modifications package to the client (See Chapter 7.2), the Sync element is required to include all data from the source database.
3. The client MUST store all data items to its database (i.e., overwrites old data) and the client MUST return the map items for all stored data items back to the server.

7.6 Error Cases

In this chapter, the recommended behaviors of devices are defined in the cases of different error types.

7.6.1 No Packages from Server

If the client has sent the empty sync command to the server, it does not get any complete response to it (new modifications), the client SHOULD drop the SyncML session and try to get the modifications later by starting the sync from the beginning.

7.6.2 No Data Update Status from Client

See Chapter 5.6.2.

7.6.3 No Map Ack from Server

See Chapter 5.6.3